IEEE Open Journal of
Intelligent Transportation
Systems

Received 20 February 2023; revised 15 May 2023 and 1 August 2023; accepted 7 November 2023. Date of publication 9 November 2023; date of current version 29 November 2023.

Digital Object Identifier 10.1109/OJITS.2023.3331449

A Survey on the Use of Container Technologies in Autonomous Driving and the Case of BeIntelli

BENJAMIN ACAR, MARC GUERREIRO AUGUSTO, MARIUS STERLING, FIKRET SIVRIKAYA, AND SAHIN ALBAYRAK

Chair of Agent Technology, Technische Universität Berlin, 10623 Berlin, Germany

CORRESPONDING AUTHOR: B. ACAR (e-mail: benjamin.acar@dai-labor.de)

This work was supported by the German Federal Ministry for Digital and Transport (BMDV) through BeIntelli Project under Grant 01MM20004.

ABSTRACT The application of containerization technology has seen a significant increase in popularity in recent years, both in the business and scientific sectors. In particular, the ability to create portable applications that can be deployed on different machines has become a valuable asset. Autonomous driving has embraced this technology, as it offers a wide range of potential applications, including the operation of autonomous vehicles and the digitization of infrastructure for the development of Cooperative, Connected, and Automated Mobility (CCAM) services. This paper provides a comprehensive analysis of containerization in autonomous driving, emphasizing its application, utility, benefits, and limitations.

INDEX TERMS Docker, containerization, automotive, CCAM, autonomous driving.

I. INTRODUCTION

INTELLIGENT systems have been disrupting many domains such as disease detection in pathology [1], speech recognition tools [2], and autonomous driving (AD) [3]. In particular, the use of Artificial Intelligence (AI) and Machine Learning (ML) techniques is taking center stage in the development of Intelligent Transport Systems (ITS) and autonomous mobility solutions. ITS enhances the functionalities of self-driving vehicles by providing additional value through the interaction with the edge (referring to road-side units) and cloud computing resources (i.e., GPU, CPU) [4], [5]. The digitized infrastructure can assist intelligent vehicles with a range of capabilities, including perception [6], routing [7], control [8], warning systems [9], and communication modules [10]. The application of these concepts is extensive, encompassing a range of applications such as parking lot availability predictions [11], object detection, Green Light Optimal System Advisory (GLOSA) [12], just to name a few examples. The integration of AI and ML techniques into ITS can enable autonomous vehicles to conclude real-time decisions, which in turn improves the safety and efficiency of the vehicle's motion. As an overarching component, the use of cloud computing resources can provide the means to support the execution and training of complex ML algorithms, such as deep learning models, which helps to improve the accuracy and robustness of ITS.

The underlying technologies, such as the software tools and programming languages used, are increasing in number and variety, including Robot Operating System (ROS) [8], Python, Kafka [13], C++, among others. With the ever-growing complexity of autonomous driving systems, the question arises of how a software stack can respond to AD, that executes diverse solutions simultaneously and without interference. Additionally, such systems are desired to be moderately maintainable and highly automated with little time from development to deployment. Against this background, we identify the subsequent key challenges for the development and deployment of autonomous driving solutions:

1) managing a complex software stack with multiple programming languages, libraries, and frameworks, while supervising and monitoring compatibility issues;
2) providing scalability and consistency across development, test, and deployment environments, critical to maintaining software quality and performance;

The review of this article was arranged by Associate Editor Goncalo Correia.

3) enabling rapid deployment and efficient version control to meet the fast-paced nature of the industry and the need for continuous integration and delivery;
4) promoting seamless integration with other systems and technologies, which is essential for the creation of an interconnected ecosystem of hardware and software components.

These challenges underscore the need for innovative solutions that streamline autonomous systems development, deployment, and maintenance while fostering collaboration and interoperability across the industry and research.

Lately, architectures consisting of microservices have been proven to fulfill these required traits [14]. Instead of developing a large integrated system - as common in the past - the system is divided into core functionalities embodying microservices that act as units in an overall system. So-called containers that address the virtualization of all components in an application have been shown to be the means of choice for realizing microservices. The use of containerization increases steadily, both in industry and science. It allows applications to be designed in such a way that these can be transferred and executed on various machines with different hardware and software characteristics. The potential applications of such a technology in the field of AD are wide-ranging, including the basic functions perception, planning and control, and also more complex concepts such as swarm optimization [15], road-side perception [16], to name just a few examples. The industry around AD already recognized the benefit of using container technologies in their products, e.g., the NVIDIA DRIVE platform [17].

In the following, we analyze the use of containerization in autonomous driving, examine where and how it is being applied, what advantages it has for the respective areas of application, and analyze its limitations. Furthermore, we review those advantages in terms of how microservice architectures can overcome the key challenges introduced above. Additionally, we compare the results for the case of the applied AD research project BeIntelli [18]. In this project, several vehicles and around 20 kilometers of road infrastructure in the heart of Berlin, Germany, were digitized with various sensors, computing resources, and communication units, in order to research and showcase connected, cooperative and autonomous mobility (CCAM) solutions.

In the remainder of this paper, Section II describes the basics of container technologies, followed by Section III, which classifies containerization use cases for AD. To emphasize container use cases, Section IV takes reference on the BeIntelli research project on autonomous mobility, which looks into the conception, implementation, test and validation of container-based connected and automated mobility services. Section V discusses the results, draws a conclusion and provides insights on future research directions and opportunities.

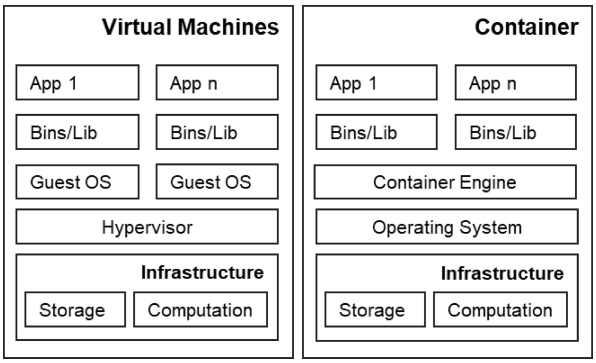


FIGURE 1. Comparison of virtual machines and containers own depiction.

II. BASICS OF CONTAINERIZATION

Numerous container runtime technologies are impacting application development activities, including Docker [19], rocket (rkt) [20], Container Runtime Interface (CRI-O) [21], container daemon (containerd) [22], and Microsoft Containers [23]. Moreover, cluster management, administration and deployment tools such as Istio [24], Kubernetes [25], Envoy [26], Apache Mesos [27] and Docker Swarm [19] help to organize DevOps for containers. This section provides a summary of the basic concepts of container technology.

A. VIRTUAL MACHINES VS. CONTAINERS

Both virtual machines (VM) and containers represent forms of virtualization. The difference between the two is the level at which the virtualization takes place. VMs realize virtualization on the hardware level. A hypervisor is employed emulateing virtual hardware such as memory, CPU, GPU, to name a few examples. A VM exists independently among other VMs and applies its own operating system (Linux, Windows etc.), on which applications can be deployed [28]. In comparison, a container is a virtualization realized at the operating system level. Every container isolates a group of resources and inhabits own processes. As no dedicated operating system is required, a container can provide more lightweight virtualization [28]. A visual comparison of virtual machines and containers can be found in Figure 1.

B. DEVOPS

One of the most important reasons for the use of container technologies is the ease and pace at which any system can be put into production. Past approaches to the deployment of applications have shown high failure rates and extensive debugging due to missing dependencies or host compatibility issues. The use of container technologies minimizes these issues as they embody self-contained units, making the transition from development to production almost seamless. These advances in the field of DevOps are one of the main reasons why container technologies is now used more often than traditional deployments [29]. Additional, the DevOps process is simplified by the provision of management and

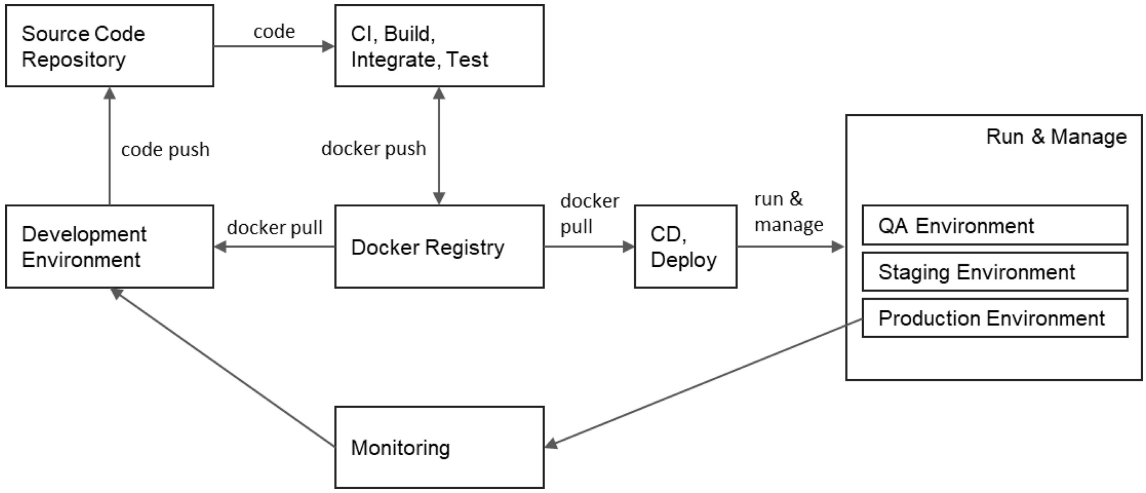


FIGURE 2. Deployment workflow in container-based architectures.

deployment tools such as Kubernetes, and orchestration tools that monitor containers, for instance, used for re-upload in the case of a failure [25]. The deployment workflow of container-based architectures is visualized in Figure 2.

C. DOCKER CONTAINER

The most widely used technology in the context of containers is Docker, launched in 2013 [19]. A Docker container can be used to deploy applications in such a manner that they operate isolated in the deployed system, with respect to deployed containers, too. The layer-wise structure enables the efficient handling of containers and tier-wise functionality management. When possible, resources can be conserved by using the layers on different images. Communication between conventional virtual machines operates similar to communication among containers. Every container has a local IP addressacting as the unique identifier for machines and containers. Although Docker containers operate independently, host directories can be mapped to provide persistence for the data they contain. The use of Docker containers is far-reaching, programming language-independent and relatively easy to set up and use, even with limited IT background. Images tailored to a vast set of requirements can be found on the official Docker Hub website [30].

III. CONTAINERIZATION IN AUTONOMOUS DRIVING

We analyze the use of containerization techniques in terms of their rationale and use cases, by conducting a literature review. First, as described in the introduction, we identified the key challenges in developing and deploying new solutions for autonomous driving. We then searched popular scholarly databases including CORE and Google Scholar for papers containing combinations of keywords such as “autonomous driving”, “container”, “docker”, “intelligent car”, as well as keywords related to the key challenges, such as “deployment” and “maintenance”. A complete list of all keywords used can be found in the Appendix. To ensure the quality and relevance of the selected literature, we established several inclusion and exclusion criteria. The inclusion criteria covered papers that provided a description of the use of container technologies, original research or case studies, as well as publications presenting new insights on whether and how containerization can address the identified key challenges for the development and deployment of autonomous driving solutions. We excluded non-English papers, those with duplicate information already covered by selected papers, and those lacking sufficient methodological and technological detail. To minimize bias in the selection and review process, at least two reviewers from the author group were involved in the analysis of each paper.

As most of the reviewed papers treat containers as mere technology enablers and focus on the general aspects of autonomous driving, they exclude details on the specific functionalities and benefits of containers in the given scenario. Therefore, in our analysis below, we enrich the literature review with additional information on the architectural features and specific functionalities that container technologies can prove to be useful in the realization of autonomous driving applications. In particular, we make a connection to the *key challenges* introduced in Section I, whenever applicable.

A. EXPLORING THE BENEFITS OF CONTAINERIZATION IN AUTONOMOUS DRIVING USE CASES

The use of containerization technology in autonomous driving applications is a topic of considerable interest, as it offers a number of properties that can be exploited to realize the use cases. In the following discussion, we systematically examine the reasons for implementing containerization in autonomous driving and highlight the specific properties that make it advantageous for these scenarios.

Lightweight: One of the most common reasons for using container technology is that containers are lightweight

virtualizations [31], [32] and are therefore resource-saving [33], [34]. This means that containers use fewer resources than traditional virtual machines, which can help improve the overall performance and stability of the host machine. This also makes containers more suitable for resource-constrained environments such as embedded systems, IoT devices, and edge computing. This property is further enhanced by the layered structure of Docker images, which allows specific layers to be used across multiple images, saving resources [33], [35], [36]. This means that the same layers, such as the operating system or shared libraries, can be shared across multiple images, helping to reduce the overall size and footprint of the images. This can help in saving storage space and bandwidth and improve the performance of the container registry. Container architectures are therefore more scalable [31], [36], [37]. Typically, connected vehicles can communicate with each other within a limited range, e.g., 250 meters [38]. This provides little time window for message exchange and coordination among vehicles moving at high speeds. Therefore, the communication system needs a quick and efficient way to process and transmit data in scenarios where connected vehicles exchange information to improve traffic efficiency and safety. The lightweight nature of containers minimizes the resource overhead and increases the efficiency of data processing and communication tasks by ensuring that only essential components are packaged and executed.

Isolation: One reason why containers are popular in the automotive domain is their intrinsic property of isolation in terms of microservice properties [33], [34], [35], [36], [39]. This means that containers provide a way to break a monolithic application into smaller, more manageable and independent services that can be developed, tested and deployed separately. This can help in improving the maintainability, scalability and overall performance of the application. Understanding the effects of vibration on point cloud data is important for autonomous vehicles, as described in [40], because vibrations from the vehicle or road conditions can affect the accuracy and reliability of the data. Developers can create algorithms that account for or compensate for the effects of vibration on LIDAR point clouds by analyzing the data from these tests. Processing LIDAR tends to be computationally intensive. By providing an efficient and isolated environment to process LIDAR data and run analytics to detect and correct for vibration-induced errors, containers can be beneficial in this scenario. This would ensure that other critical system components are not affected and that these tasks run efficiently.

Restrictions: Containers also offer the ability to constrain applications and their resource usage so that conflicts and resource bottlenecks that cause crashes and unwanted throttling on host machines can be avoided, even when multiple services are running [33]. So containers may be configured to limit their usage of resources like CPU, memory, and storage. This can help improve the overall performance and stability of the host machine by ensuring that containers do not consume more resources than necessary. In addition, this feature allows multiple services to coexist on the same machine without causing conflicts or resource bottlenecks. In developing a blockchain framework designed for smart mobility data markets (BSMD), as highlighted in [41] it is crucial to recognize the importance of efficient resource handling. This is especially important considering the size of the BSMD network, as the active nodes responsible for maintaining the ledger may need more processing and storage power. Considering that these nodes may be deployed on micro-computers, it is imperative that an effective resource-handling strategy is prioritized in order to minimize the risk of any potential outages that may occur. Underscoring the criticality of this aspect in the overall framework of blockchain technology used in smart mobility data markets, the success and functionality of the BSMD network are highly dependent on the reliability and resilience of these active nodes.

Portability: Containers are widely recognized as a technology that enables portability. The ability to package and deploy applications and dependencies as a single unit, creating a consistent and portable environment, is one of the key benefits of containerization. This enables convenient portability [31], [32], [39], as the same containerized application can be deployed on different infrastructures regardless of the underlying operating system or hardware. In addition, the flexibility provided by easy portability makes deployments more agile, allowing organizations to effortlessly adapt to changing needs and scale their applications to meet the demands of their users [34], [39]. This flexibility in turn leads to rapid development and deployment of distributed applications [32]. By packaging machine learning models and their dependencies, developers are able to deploy and maintain consistent model execution across multiple computing environments efficiently. This portability can enable vehicles to adapt more effectively to different traffic situations and environments by seamlessly integrating pedestrian behavior analysis and trajectory prediction models into autonomous driving systems [42]. In addition, it can simplify the process of updating and maintaining the models as new data and model improvements become available. This ensures that autonomous driving systems are always equipped with the latest pedestrian modeling and prediction advances.

Reproducibility: The ability to create images with all their application dependencies also makes development easier [33]. Developers can ensure that the application runs in a consistent environment, regardless of the underlying infrastructure, by packaging the application and its dependencies in a single image. This eliminates the need for manual configuration and setup, which can save developers time and effort and is particularly useful for reproducibility [43]. Using container images makes replicating the same environment across different machines effortless. This is particularly useful for test, staging, and production environments. This helps to minimize errors and improve the quality of the

final product. This feature is also beneficial for increasing the veracity of research experiment results, as it allows researchers to reproduce the same experiment in multiple environments, ensuring that the results are consistent and reliable. It is worth noting that containers have the ability to guarantee the consistency and reliability of the deployment of BSMD nodes, as shown in [41]. This can be applied to a variety of devices and environments, including different hardware platforms such as microcomputers, e.g., Raspberry Pis, as well as cloud-based or dedicated servers. Therefore, regardless of the specific device or environment, using containers is a highly effective strategy to ensure that the deployment process remains smooth and stable and that the BSMD nodes can function optimally.

Orchestration: Another advantage of container technologies is the ability to use orchestration tools to manage the containers in terms of monitoring and rescheduling in case of failure [33]. Container orchestration platforms such as Kubernetes, Docker Swarm, or Mesos provide a consistent and unified way to deploy, scale, and manage containerized applications across different infrastructures. This allows organizations to easily manage and monitor their containerized applications, ensuring they are running smoothly and efficiently. One of the key features of container orchestration platforms is automatic scaling, which allows organizations to add or remove containers to meet application demands conveniently. Running only the required containers can help improve application performance, availability and reduce cost. In addition, these platforms offer self-healing capabilities that allow them to detect and recover from failures, improving application reliability automatically. For data management systems in service-oriented testbeds, as developed in [44], the property of orchestration can be useful. More specifically, in the Data Processing Layer (DPL), which provides a way to transport data between entities in the system, usually from different sources, the orchestration feature of containers helps to ensure efficient resource allocation, load balancing, and fault tolerance across data processing tasks. Thus, such systems can dynamically adapt to the changing workload and provide better performance, reliability, and scalability in the processing of the live data streams in the testbed ecosystem.

CI/CD: The ability to leverage Continuous Integration / Continuous Delivery (CI/CD) pipelines is another reason why container technologies are particularly attractive at the production level [35], [45]. This means that containerization can be used to automate and streamline the entire software development life cycle, from writing code to testing and deploying to production. This can improve the overall quality and performance of the application by enabling faster and more reliable deployment of new features and bug fixes. Using containerization in CI/CD pipelines also enables faster and more efficient testing, as the same containerized environment can be used for development, testing, and production, ensuring consistency and reducing the risk of errors. Furthermore, the use of containerization facilitates the realization of DevOps [35], [43], which is a methodology that encourages development and operations teams to collaborate to improve the speed and quality of delivering software. In addition, containers can be easily deployed, scaled, and managed across a variety of environments, which can help to improve the agility of both development and operations teams. Overall, the use of containerization in production environments can lead to a more efficient and maintainable container architecture [43], [46], providing a number of benefits such as the use of CI/CD pipelines and the ease of DevOps realization. This can improve the speed, reliability, and overall quality of the application while reducing the risk of failure. This is particularly important for communication-intensive functions such as platooning [47]. CI/CD can ensure faster, more reliable updates with minimal downtime in platoon dynamics and communication between RSUs and connected and automated vehicles. Resulting in a more robust and stable communication system, which is essential for the efficient and safe operation of participating vehicles in a platoon.

Quality Assurance: In the field of autonomous driving, ensuring the quality of software solutions is paramount, as they are responsible for making critical driving decisions that can affect the safety of passengers and other road users. Therefore, using containerization to help ensure quality has become particularly valuable. The ability to effortlessly transfer solutions to any machine is one of the key benefits of containerization in autonomous driving. When testing and evaluating the performance of autonomous systems, this feature is especially important. With containers, solutions can be packaged with all their dependencies and configurations for convenient migration to different environments. Allowing developers and testers to simulate real-world scenarios and evaluate how the autonomous system performs under varying conditions. They can identify and resolve potential problems before they occur in the field by testing solutions in different environments. This increases the reliability and safety of the autonomous system by ensuring that it can function properly in a variety of conditions and environments. Therefore, it assures the quality of the solutions [34], [36].

Privacy Preservation: Containers are gaining popularity as a secure and privacy-preserving technology [33]. By separating the application and its dependencies from the host system, their isolation capability provides an extra layer of security. This prevents unauthorized access by limiting access to sensitive data and system resources. Encapsulating containers also makes it harder for an attacker to compromise the system. In addition, the smaller attack surface of containers compared to virtual machines makes it harder for an attacker to exploit vulnerabilities in the system. The risk of data breaches is reduced since the separation of data in containers allows for fine-grained control over access to sensitive data. All in all, containerization technology provides a secure and privacy-preserving solution for the running of applications and the protection of sensitive data. The process of autonomous driving often involves the

collection of data from a variety of sensors, including but not limited to the Global Positioning System (GPS) and Electronic Stability Control, as described in [48]. Given the sheer size and complexity of this data collection process, it is of paramount importance to protect these sensors from unauthorized access by other software programs operating within the vehicle, or by external entities that may threaten the security and integrity of the data. Underscoring the criticality of this aspect in the overall autonomous driving framework. Failure to prioritize the protection of these sensors can lead to unwanted interference and potentially disastrous results.

Time Conservation: Containers are widely recognized as a technique that can increase productivity and save time. Containerization's ability to speed up the development and deployment process is one of its key benefits. Through the use of containers, developers can package and deploy their programs and dependencies as a single unit, reducing the need for manual configuration and setup, e.g., regarding infrastructure or vehicle hardware and system software. For developers, this can be a huge saving in terms of time and effort [33]. It becomes extremely important to prioritize time savings during the development phase of novel solutions when it comes to intricate undertakings, such as the sensor fusion process described in [49]. This is due to the fact that the complexity of such tasks can lead to prolonged development duration, resulting in delayed results and increased resource utilization, both of which can have a detrimental effect on the overall productivity and profitability of the project. Therefore, in order to effectively streamline the development process and optimize output, it is imperative to place a high level of importance on saving time.

Interoperable Application Scheduling: Because of their ability to package applications and dependencies together and deploy them as a single unit, containers allow the same application to be deployed on different infrastructures, regardless of the underlying operating system or hardware. This interoperability allows applications to be easily scaled and managed across environments [33]. This is important for software-in-the-loop approaches to automated driving, where multiple simulation units and a variety of different modules, e.g., for signal conversion [48], are used. Especially when running on different platforms and environments, managing these components and their interactions can be complex and error-prone.

Finally, the accessibility of Docker container technology to developers with no previous experience in containerization [43] has led to its widespread adoption in recent years. This is largely due to the simple concepts used in the technology, allowing for a relatively low barrier to get started. Docker makes it easy for developers to create and manage isolated environments for their applications by providing a user-friendly interface for managing and deploying containers. The *time conservation* aspect of containers, as covered earlier in this section, addresses the *key challenge 3* for autonomous driving solutions, i.e., rapid deployment. DevOps practices are also facilitated, as described in the *CI/CD* section. *Reproducibility* features provide consistency across different environments, ensured by the *interoperable application scheduling* and *portability* features of container technologies. In addition, their *lightweight* nature enables scalability and supports higher software quality and performance, addressing *key challenge 2*.

On the other hand, containerization has some drawbacks, such as increased complexity, security concerns, delays in the process of creating containers, and limitations in the ability to run stateful applications. [50], [51], [52] highlight the challenges developers may face when using containers in production environments. In Section IV-B, we also present the issues and pitfalls based on our own experience of designing and deploying the container-based architecture in the BeIntelli research project.

B. USE-CASES OF CONTAINERIZATION IN AUTONOMOUS DRIVING

As the field of autonomous driving continues to evolve, containerization has emerged as a viable solution for a variety of use cases. Here, in contrast to the previous section, we explore the specific applications of containerization in autonomous driving. In the current literature on containerization, there is a notable tendency to use the terms 'Docker' and 'containers' interchangeably. This can be attributed to the prominent position that Docker currently occupies in the container technology landscape [53]. Notably, our analysis of the entirety of the references in question did not reveal any explicit mention of alternative container technologies other than Docker. The exact methodology for deploying the following applications in containers and the number of different containers used to create these deployments cannot be definitively determined without access to the corresponding container images. However, given the malleability inherent in containerization, it is reasonable to postulate that the entirety of the code is deployed within the container images. In particular, in scenarios where a single entity serves as the embodiment of the application, it is likely that all the required components are contained within a single container image.

Communication: Reference [35] uses container technologies to investigate the use of *Data Distribution Services*, a data-centric publisher-subscriber middleware standard enabling service-2-service and service-2-vehicle communication. By sharing middleware between services via image layers, [35] presents an approach for developing and operating future automotive software architectures. Real-Time Publish-Subscribe-UDP is used to communicate and OpenDDS to distribute data. The developed solution is tested on a testbed, which consists of several Raspberry Pis.

Detection: Reference [34] uses Docker to implement real-time 360° tracking and 3D multi-object detection. It records the trajectories of objects and predicts their future paths to avoid collisions and other dangerous incidents. The communication interface is ROS [8]. The developed

pipeline uses fast encoders for object detection, 3D Kalman filter, and Hungarian algorithm for state estimation and data association. The pipelines are validated in simulation environments using CARLA [54] and in the real world using an autonomous electric car equipped with NVIDIA AGX Xavier [55].

Data Generation: Reference [37] presents an automated pipeline for the automatic generation of city maps for use in simulations, where each step of the pipeline is implemented in its own Docker container. With a realistic benchmark, the new maps can then be used to test autonomous driving features. The maps are generated based on OpenStreetMap [56]. The geometry is created with Blender [57], producing FBX files which can be used in the city generation process. OpenDRIVE is used as the format specification [58]. Netconvert [59], which is part of the SUMO package [60], is used to extract the OpenDRIVE file. The generated data is used for simulation environments, in particular, CARLA [54].

DevOps: Reference [35] specifically focuses on implementing a service-oriented architecture (SOA) for the automotive industry to meet DevOps standards. The authors argue that the fast-paced and highly dynamic environment of the industry does not lend itself well to traditional automotive software development processes. To address this, they propose a SOA. The goal is to improve collaboration and communication between different teams and departments, in order to increase flexibility and scalability.

Autonomous Driving Stack (ADS): Reference [36] has developed an ADS that is divided into layers. Each layer implements different functionalities of the autonomous car, such as perception and control. The technologies used are Docker, ROS [8], and CARLA [54]. The applied concepts include HD Maps [61], RANSAC-3D algorithm [62], KD Tree [63], Fast Library for Approximate Nearest Neighbor (FLANN) [64], Hierarchical Interpreted Binary Petri Nets (HIBPNs) [65], and Linear-Quadratic Regulator (LQR) [66]. CARLA [54] is used to validate the results in simulation environments. Reference [45] is developing a cloud computing platform based on container technologies for simulating autonomous driving. CarMaker is the technology used. As a format specification, OpenDrive is applied [58]. The results are evolved in the simulation environments.

Edge Computing: Reference [33] focuses on a software stack that runs on edge computing units. The authors aim to develop an offloading decision engine, an offloading scheduler, and edge offloading middleware to be deployed with containers at the edge. The applied concepts include Multidimensional Knapsack Problem (MMKP) [67], Greedy Algorithm (GA) [68], Random Algorithms (RA) [69]. Results are validated on real-world edge computing systems. Reference [31] uses Docker containers to deploy applications in the context of connected and autonomous vehicles at the edge of the network. This is done under the Multi-access Edge Computing (MEC) paradigm. With such an approach, basic functionalities such as object recognition can be outsourced to an edge computer, instead of running on the car's computing unit. The technologies used include 5G, srsLTE [70], 5G-EmPOWER Software-Defined Radio Access Network (SD-RAN) [71], Evolved Packet Core (EPC) [72], nextEPC [73], Open vSwitch [74]. A toy car is used to validate the experiments in a real-world scenario. Reference [32] uses containerization to transfer services between different edges. This way, the service can always be delivered to the moving car with the lowest possible latency. The technologies used include 5G Radio Access Network (RAN) [75], Cellular Vehicle-to-Everything (C-V2X) [32], PC5 Sidelink Interface [76], ETSI Mobile Edge Computing (MEC) [77], ME Orchestrator [78]. The results are emulated on two real MEC devices.

Testing: Reference [43] implements software-in-the-loop testing using containers to ensure the reliability of systems and the codebase to tackle the problems of competitor programs is often difficult to install, intricate to update, and not always suitable in terms of complexity. The used technologies include ROS [8], MATLAB, Simulink and code of the CIRCLES project [79]. The results are validated in simulation environments. Reference [39] use containerization to validate a fully-autonomous driving architecture. The used technologies encompass CARLA [54], ROS [8] and the RoboGraph tool [65]. The concepts applied include Hierarchical Interpreted Binary Petri Nets (HIBPN) [65], Pure Pursuit algorithm [80], A-Star algorithm [81], Beam Curvature method (BCM) [82], Efficient Residual Factorized ConvNet (ERFNet) [83], Precision Tracker approach [84], BEV Kalman Filter [85], Nearest Neighbour algorithm [86]. The results are validated in simulation environments, using CARLA [54].

Recent research shows that the use cases for containers are far-reaching, ranging from simulation environments to real-world applications, and can help develop and deploy new autonomous driving solutions. The variety of technologies (ROS, CARLA, Edge Computing, Simulink, etc.) used for realizing the described use cases and the interplay of those containerized components show that *key challenge 4* can be overcome using container technologies. Therefore, integration with other systems and technologies is given, resulting in an interconnected ecosystem of hardware and software components.

C. COMPARISON TO VIRTUAL MACHINES

As mentioned in Section II-A, virtual machines and containers differ in their degree of virtualization. Several studies have compared these virtualization concepts and identified their relative advantages and drawbacks, which we briefly discuss in this section. Generally, containers scale better compared to virtual machines [87]. In terms of storage and memory, virtual machines utilize a considerable amount of space and resources on the host machine, while containers have a more dynamic nature in memory allocation and require less memory [88]. Furthermore, virtual machines have a (slightly) higher overhead compared to containers in

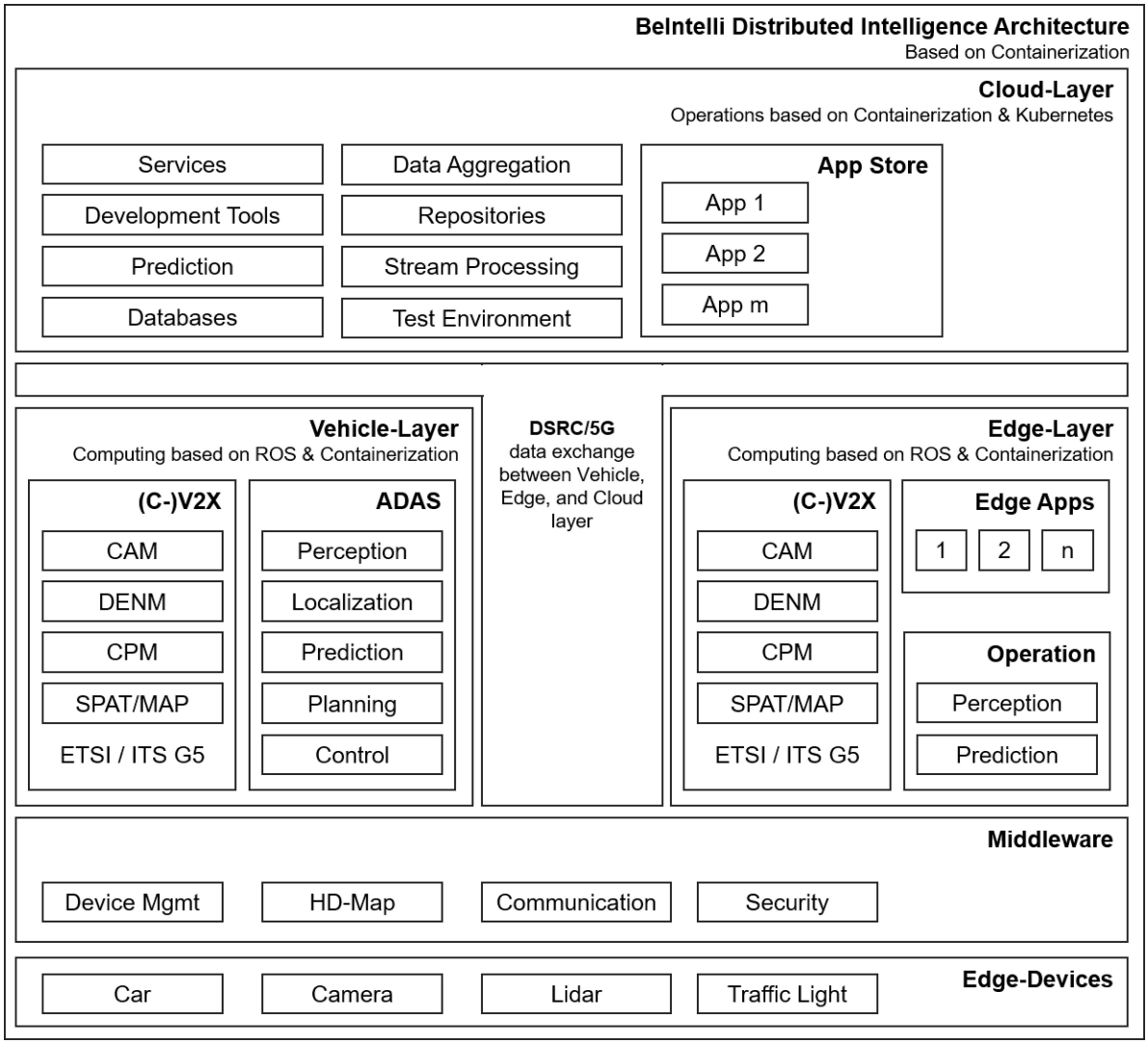


FIGURE 3. BeIntelli 3-layer architecture, consisting of vehicle, edge and cloud, communication, and respective data exchange.

terms of performance and efficiency [88]. Even with access to physical memory, virtual machines do not perform as consistently as containers [87]. The host swaps memory between disk and physical memory when the number of links exceeds the capacity of the server. However, there are certain aspects in which virtual machines are advantageous. As stated by [88], virtual machines offer a slight security advantage over containers due to their isolation. This property plays a crucial role if additional assistant services, such as GLOSA on the vehicle's on-board computer, are executed. Deploying such optional, less important functionalities alongside crucial driving functionalities, e.g., vehicle control and perception, makes the driving system more vulnerable and prone to interference by these functionalities. Containers can not provide the necessary isolation, while virtual machines provide additional security. In addition, virtual machines outperform containers in cryptographic testing [88]. Since much information in the context of autonomous driving is sensitive, for example, the GPS signal, cryptographic approaches are already integrated into the field [89]. Therefore, saving time in applying these concepts is beneficial in itself.

IV. BEINTELLI

In this section, we introduce our research project *BeIntelli* - Showcase for AI in Mobility based on Platform Economy [18]. Afterward, we analyze the project in terms of containerization aspects. The layer-based structure of our software stack is visualized in Figure 3.

A. APPROACH TO AUTONOMOUS DRIVING

Advances in the field of AI fostered applications such as autonomous driving and complementing use cases such as GLOSA [90], Smart Parking [91], Vehicle-on-Demand [92], navigation or routing. The ratio of smart and digitized vehicles steadily increases. While large automotive companies such as Mercedes-Benz, Tesla, and BMW are developing autonomous driving functions, particularly on

the vehicle level, the expected digitization of infrastructure by components such as sensors (camera, LIDAR) and edge computing units, would add to existing approaches of autonomous driving. First attempts of such systems have already been made in several projects across the globe [4]. The BeIntelli research project led by the DAI Laboratory of the Technische Universität Berlin, Germany [18] follows a threefold approach to autonomous mobility, consisting of the layers:

- *Vehicle:* Consisting of a multi-sensor setup, communication, and computing units
- *Edge:* Areas defined by roadside units with dedicated sensor setup, communication, and computing unit
- *Cloud:* System to provide aggregated data and predictions, and computing unit

The instances Vehicle, Edge and Cloud communicate, interact and assist each other and thus enable CCAM solutions. This so-called distributed intelligence approach extends the vehicle's perception to areas that would otherwise remain unreached, which we call collaborative perception. Road-side perception is realized by edge computing to which mostly cameras, road-condition sensors, and LIDARs are linked or directly attached. This data is analyzed in real-time on the edge, processed, and converted into assistance information that provides added value for a vehicle's decision-making. Through this, vehicle movements can be optimized, safety can be enhanced, and integrated services can be developed.

Likewise information for non-driving essential services can be provided to the edge or cloud, collected, processed, and provided as a service to vehicles or drivers' smartphones, e.g., the occupancy status of parking spots for a smart parking service.

B. CONTAINERIZATION IN BEINTELLI

The BeIntelli project extensively utilizes containerization through Docker containers, with the majority of services deployed as microservices. This approach is particularly common in edge computing and cloud environments, in which a global Kubernetes cluster orchestrates all applications. In such a setup Docker containers are commonly used as the virtualization for Kubernetes [25]. The vehicle itself is included in the Kubernetes cluster, but due to security reasons and to achieve unfettered access not all software components are included. For security, an additional on-board computer is used to separate services that are not directly related to controlling, perception and routing the vehicle from the rest of the system. This ensures that even in the event of a total failure of the Kubernetes cluster, the vehicle remains operational. Moreover, this approach allows the exclusion of potential sources of interference with critical functionalities like real-time control from additional services like GLOSA [90]. Services at the cloud level, such as parking prediction [93] or congestion assistant, are all deployed in containers as microservices. Services running at the edge are also packaged in containers and controlled by Kubernetes orchestration, such as object detection for perception [94]. The same goes for the vehicles - the on-board computer for running non-critical services is orchestrated by Kubernetes, and the on-board computer for control, perception, etc. is not while all services are containerized. To do this, the underlying ROS used [8] is also containerized. Furthermore, the communication realized by the Kafka publish and subscribe technology [13] is deployed in containers. Thus, instead of pushing information directly to a Kafka broker to communicate, the information is pushed to a container that contains a Kafka broker. The communication design in this project uses Kafka brokers both at the edge and cloud level. This allows for low-latency conversion of edge-generated data into messages. It also provides a central communication hub from which data can be retrieved. For additional redundancy and data protection, data generated at the edge is also forwarded to the cloud Kafka broker using forwarding scripts. Thus, on all levels entities are implemented in containers.

A platform to facilitate the development, deployment and testing of CCAM solutions by various stakeholders, including companies, start-ups and academia, will be developed in the context of the BeIntelli project. This requires implementing a pipeline for easy and seamless deployment of solutions from local machine to the vehicle, infrastructure, and cloud without requiring extensive access to the underlying infrastructure and knowledge about it. Docker containers are particularly well suited for this purpose due to their ease of portability and ease of use. In addition, Docker containers' widespread popularity among developers, due to its steady growth over recent years, renders it a familiar technology that is embraced by a wide range of experts from different fields. This facilitates collaboration and straightforward and low-threshold solution development in the autonomous driving space. The experience gained in the predecessor project of the BeIntelli project suggests that while serverless approaches are intriguing, they have not yet gained widespread acceptance among developers. As a result, architectures based on them are difficult to maintain.

In the BeIntelli project, the use of orchestration tools such as Kubernetes is essential. This is due to the large infrastructure and the large number of applications running on numerous edge computer and different sensor types. Therefore, deployments in the traditional way are difficult and require extensive hardware adaption. As indicated, the hardware is heterogeneous: The first part of the testbed was built during the predecessor project, Diginet-PS [95], and the BeIntelli project extends it in a second phase. The experience gained during the first digitization phase and technological advances have influenced the choice of hardware and components for the second digitization phase. The different hardware must be considered while ensuring software compatibility across all devices.

The exploitation of tools like Kubernetes has further advantages besides the mentioned beneficial core feature. To manage communication between various system components

and external services, e.g., cloud-based data storage or real-time traffic updates, customizable networking is essential in the context of autonomous driving. For example, by optimizing data transfer and reducing latency for time-critical operations, such as sensor fusion or (collaborative) perception, the networking options provided by Kubernetes allow developers to design a network architecture that meets the unique requirements of their autonomous driving system. This is done by leveraging the Kubernetes namespace and networking policies. Autonomous driving systems have complex configurations with many parameters including sensor calibrations, control algorithms, and communication protocols among others. Kubernetes *ConfigMaps* and *Secrets* objects allow developers to manage these configurations more efficiently by keeping them in a central location. This makes it easier to track and versioning configurations, and ensure consistency throughout the development lifecycle. With this setup developers can easily switch between different configurations of for example camera calibration parameters by switching to different *ConfigMaps* in order to test and compare performance in different scenarios.

Regarding portability, containerization is critical for seamlessly integrating autonomous driving solutions into different vehicle models and hardware platforms. For example, without having to customize the software stack, an autonomous driving system developed for one of the cars can be adapted and deployed in other vehicles by migrating the Docker container. This is especially important for software solutions that are supposed to be adapted for different vehicles or vehicle types. In the case of the BeIntelli project, there are a variety of vehicle types, such as a car, minibus, SUV, and bus, all using the same software stack. As the employed autonomous driving solutions evolve and become more sophisticated, they may also require more processing power and storage capacity, for example, due to more complex sensor setup with more cameras and LIDARs in the bus in contrast to cars. Containers' scalability ensures that our growing needs can be met without significant reconfiguration of the underlying infrastructure. And because the BeIntelli project is a research project, it requires testing and evaluating, for which containers allow quickly deploying updates, such as refined algorithms for object recognition or path planning, to accelerate development and research.

Another advantage is the container ecosystem, more precisely, of Docker. For example, many out-of-the-box images are already available such as the map handling library Lanelet2 [96], which reduces development time and effort. As described above, all our applications and units are deployed in containers. Therefore we explicitly use the modularity principle of container technology to make our system more robust against shutdowns, less complex, and more maintainable. Incremental updates also save us significant time and resources during the development process. For example, developers can update only the affected layer within the Docker container instead of rebuilding the entire autonomous driving system when a change is made to the vehicle's control. This allows us to iterate faster and deploy improvements and bug fixes more efficiently, ultimately accelerating the development of safer and more robust autonomous driving solutions. Since the complete on-vehicle sensor recordings are huge (about 1.6 TB per 1h drive), this is particularly important.

As shown, container technology has many advantages. However, it also has some disadvantages. One such drawback is the opaque versioning of code that can occur with container technology. In practice, the code running in a Docker container may not be able to be accurately correlated to the appropriate code repository, leading to confusion and difficulty when changes are needed. This is especially problematic for organizations with large development teams and complex codebases. It can be difficult to determine which version of code is running in production and which version of code is being worked on in development without a clear correspondence between the code running in the container and the specific code repository. Our solution to this problem is the implementation of a system that uniquely identifies the correspondence between code and specific repositories, as well as the state of the repository. This is very useful in large environments because it streamlines development processes and reduces confusion. By adding configuration files to all Docker containers that outline the specific version of code running in each container, the system provides automated tools for versioning and tagging code. This allows developers to easily identify the appropriate code repository and version when changes are needed, and it also makes it easier to roll back to previous versions of code when necessary. It also improves security and compliance in large environments by providing a clear and accurate record of the code running in each container. It helps ensure that only approved versions of code are running in production. It also makes it easier to identify and fix any vulnerabilities that may exist in the code. Therefore, the *key challenge 3*, in terms of version control, can not be met by default through container technology. However, with small effort, this feature can be implemented as well.

The expectation that only a single main process should run inside each container is another limitation of Docker containers. While the intent of this design principle is to enforce application modularization, it can also be a constraint limiting application flexibility in certain cases. For example, if an application requires multiple main processes to run concurrently, it must create multiple containers. This can lead to increased resource consumption and management complexity. To our knowledge, using a supervisor tool is the only way to work around this limitation which requires the creation of a complex configuration file. This would need to be created and maintained manually and could be prone to errors, causing the application to behave unexpectedly. In addition, this approach may add an extra layer of complexity to the application, making it more difficult to understand and maintain. However, at this time, without using workarounds such as the supervisor daemon to run multiple processes

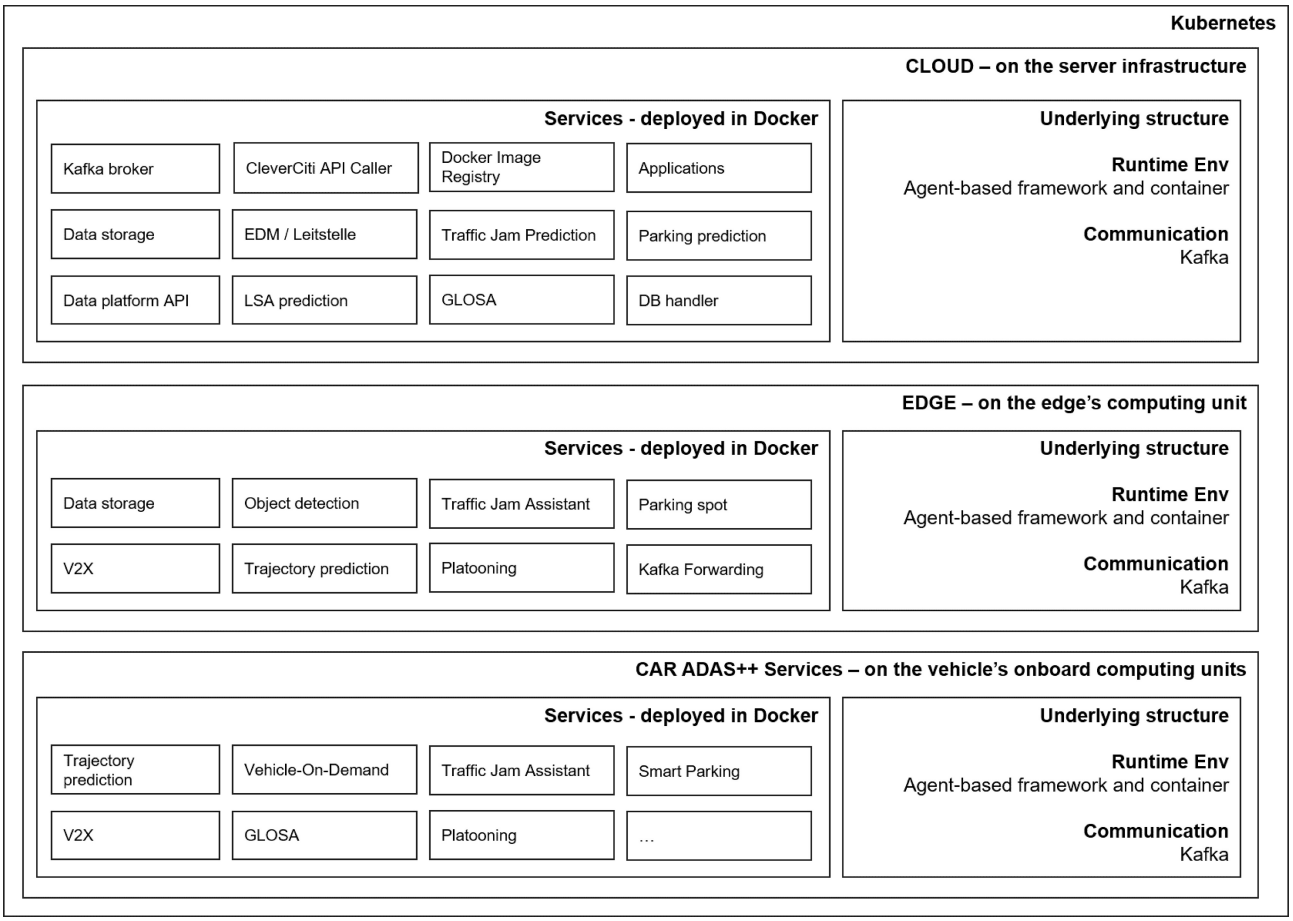


FIGURE 4. Applications of container technology in the research project BeIntelli.

within a container, we are not aware of a straightforward method for starting multiple main processes simultaneously within a single container.

Efficiently managing and distributing Docker containers to edge computers is another challenge. It is critical to optimize the distribution of container layers to edge devices based on predictive analytics, as the deployment of autonomous vehicles relies on real-time processing and decision-making. However, it proved to be a complex task to accurately predict the requirements and efficiently transfer the necessary layers in advance to save time when deploying those whenever required. Since edge computers are limited in terms of computational power, storing all container images on every machine is not a valid approach. Therefore, extensive planning and coordination is required to ensure that the right layers are available in the right place at the right time.

Orchestrating for infrastructure-enhanced autonomous driving is also difficult. Kubernetes is a widely used orchestration tool, though it was not designed specifically for the unique requirements of autonomous driving use cases. Our project includes digitized infrastructure components like cameras that provide over-the-horizon perception capabilities. The integration of these digital assets with traditional orchestration frameworks is a challenge, for instance because of the limited resources on each edge computer. Thus, an orchestration tool was required that could efficiently leverage the digitized infrastructure while taking into account the specific needs of autonomous driving, such as real-time sensor data processing and coordination with vehicle control systems.

In summary, despite certain limitations, the use of container technology has proven to be a viable solution for the BeIntelli project. Container technology's advantages, such as improved scalability and reduced complexity, outweigh its disadvantages, like opaque versioning and limited number of main processes. A large number of applications have been successfully deployed and orchestrated within the project's extensive infrastructure through the use of container technology. These applications are developed in different languages (Java, Python, HTML/CSS, JavaScript etc.), use different libraries (React, TensorFlow etc.) and are deployed on different virtual and physical machines. Therefore our project confirms that container technology overcomes the *key challenge 1* in developing and managing a complex software stack. Overall, the decision to adopt container technology was sound. A visualization of our container architecture is provided in Figure 4.

V. DISCUSSION & CONCLUSION

With the advent of autonomous driving several key challenges emerged that must be addressed to realize its full potential. These challenges include a complex software stack with multiple programming languages and libraries, the need for scalability and consistency across different units, the need for rapid deployment coupled with efficient version control, and seamless integration with other systems. Container technologies have emerged as powerful tools that mitigate many of these challenges, improving the overall development and management of autonomous driving systems. Docker's containerization approach greatly simplifies the management of complex software stacks by encapsulating dependencies and isolating execution environments. This reduces the complexity associated with the integration of different programming languages, libraries, and other components, and also streamlines the development process and ensures that developers can focus on their core tasks without being bogged down by compatibility issues. Additionally, containers' lightweight nature enables easy scalability and improved consistency across entities. Regardless of the underlying infrastructure, this ensures that the performance and behavior of autonomous systems remain stable and predictable. Containers promote rapid deployment of these systems while minimizing inconsistencies arising from different execution contexts by providing a unified and standardized environment. In spite of the many benefits that container technologies offer, it is important to recognize that they do not provide a robust version control system by themselves. However, container technologies play an important role in facilitating seamless integration with other system, which is especially crucial in the context of autonomous driving, where integrating various sensors, communication systems, and other subsystems is essential to achieve desired functionality and safety. Containers greatly simplify the integration process and ensure that different subsystems work together harmoniously by providing a standardized environment that can accommodate diverse components.

The BeIntelli project, which utilizes Docker containers extensively, is a prime example of containerization technology's advantages and effectiveness in infrastructure-enhanced autonomous driving. Using Docker containers and Kubernetes orchestration allows for friendly deployment, testing, and compatibility across various hard- and software stacks while ensuring the separation of crucial functionalities and protecting against total system failure. Additionally, the communication in the project is also implemented in containers, using especially Kafka publish-and-subscriber technology. The BeIntelli project also aims to develop a platform that facilitates the development, deployment, and testing of CCAM solutions by various stakeholders, using Docker containers as the foundation for our pipeline. Overall, the research suggests that containerization technology is a crucial enabler for the successful implementation of autonomous driving solutions, nowadays and in future.

VI. FUTURE WORK

In the following, we provide insights on possible future research directions based on our work.

A. PREDICTIVE LAYER TRANSFER

As more applications require software to be deployed across multiple nodes or machines, including infrastructure-enhanced autonomous driving, the use of distributed systems has become increasingly common in modern computing. To achieve optimal performance and scalability in such systems, the efficient transfer of software, data, and other resources is essential. Service demand predictions in distributed systems is one promising area [97]. Software can be transferred to the required locations in a timely manner, reducing delays and improving overall performance, by predicting the demand for a specific service. However, the potential benefits of predicting in distributed systems are not limited to moving complete services through the network. There is also an opportunity to predict demand for specific software layers in the context of Docker-based systems. By analyzing usage and demand patterns and identifying the most effective predictive variables, this can be accomplished. These layers can be moved to different registry locations for use as needed by predicting the demand for specific layers. The benefits of being able to predict the demand for specific layers in Docker-based systems might be significant. The prediction model can be refined and fine-tuned for optimal results by optimizing the layer structure. A number of factors must be considered, including the size and complexity of the layers, the frequency of use, and the degree of interdependency between layers. How the prediction should be performed to achieve the best possible accuracy is also an important consideration. This can include using various machine learning techniques and algorithms to analyze patterns in usage and demand. For example, one might use clustering to identify groups of layers with common usage, or regression to predict demand for specific layers based on historical usage data. Another important question is whether there are differences between the prediction of demand for specific layers and the prediction of demand for complete container images or software packages. This is an area that needs further investigation. However, it is likely that predictive models would need to be adjusted depending on the level of granularity required. For example, prediction of demand for specific layers may require a more granular approach, while prediction of demand for full images may be better suited to a broader, more holistic approach.

B. OPTIMIZED ORCHESTRATION FOR INFRASTRUCTURE-ENHANCED AUTONOMOUS DRIVING

A variety of research questions and opportunities arise from exploring the application of orchestration tools for infrastructure-enhanced autonomous driving. How orchestration paradigms may need to be adapted or extended to meet the unique requirements of this specific use case is a fundamental aspect to be explored. We should examine

the different orchestration strategies required to manage the complex interactions between autonomous vehicles and digitized infrastructure components, and the tradeoffs arising from such adjustments. Key research questions that need to be addressed may include the following:

1) How can the orchestrator dynamically assign and manage resources based on the changing needs of autonomous vehicles and infrastructure components to provide optimal performance?
2) To create a flexible and efficient hybrid infrastructure, what strategies can be employed to facilitate seamless orchestration across different cloud providers, edge devices, and on-premises systems?
3) How can we develop intelligent policies for the orchestration system that can adapt to changing conditions such as network congestion, hardware failures, or changing computing needs?

Challenges related to scalability, real-time computing, security, edge computing, fault tolerance, network optimization, and resource management must be addressed to optimize orchestration for a testbed with digitized road infrastructure and autonomous vehicles. We should research on a more efficient and robust orchestration system that can effectively manage the complex interactions between different components in this highly dynamic environment by exploring research questions in areas such as dynamic resource allocation, cross-platform orchestration, intelligent policies, energy efficiency, predictive resource management, interoperability, and monitoring.

APPENDIX

The keywords used for the literature research in this article are (in alphabetical order): Autonomous Driving, CARLA, Communication, Container, Data, Data Acquisition, Deployment, DevOps, Docker, Edge, Edge Computing, Fog, Fog Computing, Intelligent Car, Maintenance, Software, SUMO, Test Environment, Testing.

ACKNOWLEDGMENT

Furthermore, the authors acknowledge support by the German Research Foundation and the Open Access Publication Fund of TU Berlin.

REFERENCES

[1] M. Y. Lu et al., "AI-based pathology predicts origins for cancers of unknown primary," *Nature*, vol. 594, no. 7861, pp. 106–110, 2021.
[2] S. K. Gaikwad, B. W. Gawali, and P. Yannawar, "A review on speech recognition technique," *Int. J. Comput. Appl.*, vol. 10, no. 3, pp. 16–24, 2010.
[3] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE Access*, vol. 8, pp. 58443–58469, 2020.
[4] M. Augusto, A. Hessler, J. Keiser, N. Masuch, and S. Albayrak, "Towards intelligent infrastructures and AI-driven platform ecosystems for connected and automated mobility solutions," in *Proc. ITS World Congr.*, vol. 27, 2021, pp. 2364–2373.
[5] Z. Bai, G. Wu, X. Qi, Y. Liu, K. Oguchi, and M. J. Barth, "Infrastructure-based object detection and tracking for cooperative driving automation: A survey," in *Proc. IEEE Intell. Veh. Symp.*, 2022, pp. 1366–1373.
[6] D. Gruyer, V. Magnier, K. Hamdi, L. Claussmann, O. Orfila, and A. Rakotonirainy, "Perception, information processing and modeling: Critical stages for autonomous driving applications," *Annu. Rev. Control*, vol. 44, pp. 323–341, Oct. 2017.
[7] A. Bhat, S. Aoki, and R. Rajkumar, "Tools and methodologies for autonomous driving systems," *Proc. IEEE*, vol. 106, no. 9, pp. 1700–1716, Sep. 2018.
[8] A. Koubâa et al., *Robot Operating System (ROS)*, vol. 1. Cham, Switzerland: Springer, 2017.
[9] J. D. Rupp and A. G. King, "Autonomous driving—A practical roadmap," SAE, Warrendale, PA, USA, Technical Paper, 2010.
[10] J. Wang, J. Liu, and N. Kato, "Networking and communications in autonomous driving: A survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 1243–1274, 2nd Quart., 2018.
[11] Y. Zheng, S. Rajasegarar, and C. Leckie, "Parking availability prediction for sensor-enabled car parks in smart cities," in *Proc. IEEE Tenth Int. Conf. Intell. Sens., Sens. Netw. Inf. Process. (ISSNIP)*, 2015, pp. 1–6.
[12] B. Bradaï, A Garnault, V. Picron, and P. Gougeon, "A green light optimal speed advisor for reduced CO2 emissions," in *Energy Consumption and Autonomous Driving*. Cham, Switzerland: Springer, 2016, pp. 141–151.
[13] "Kafka," Accessed: Oct. 14, 2022. [Online]. Available: https://kafka.apache.org/
[14] E. Wolff, *Microservices: Grundlagen Flexibler Softwarearchitekturen*. Heidelberg, Germany: Dpunkt.verlag, 2018.
[15] S. Kachroudi, M. Grossard, and N. Abroug, "Predictive driving guidance of full electric vehicles using particle swarm optimization," *IEEE Trans. Veh. Technol.*, vol. 61, no. 9, pp. 3909–3919, Nov. 2012.
[16] R. Yu, D. Yang, and H. Zhang, "Edge-assisted collaborative perception in autonomous driving: A reflection on communication design," in *Proc. IEEE/ACM Symp. Edge Comput. (SEC)*, 2021, pp. 371–375.
[17] "Nvidia drive platform," Accessed: Apr. 26, 2023. [Online]. Available: https://developer.nvidia.com/drive/docker-containers
[18] "Beintelli," Accessed: Oct. 15, 2022. [Online]. Available: https://beintelli.com/
[19] "Docker," Accessed: Oct. 14, 2022. [Online]. Available: https://www.docker.com/
[20] "RKT," Accessed: Nov. 20, 2022. [Online]. Available: https://www.redhat.com/
[21] "CRI-O," Accessed: Nov. 20, 2022. [Online]. Available: https://cri-o.io/
[22] "Containerd," Accessed: Nov. 20, 2022. [Online]. Available: https://containerd.io/
[23] "Windows container," Accessed: Nov. 20, 2022. [Online]. Available: https://learn.microsoft.com/
[24] "Istio," Accessed: Nov. 20, 2022. [Online]. Available: https://istio.io/
[25] "Kubernetes," Accessed: Oct. 15, 2022. [Online]. Available: https://kubernetes.io/de/
[26] "Envoy," Accessed: Nov. 20, 2022. [Online]. Available: https://www.envoyproxy.io/
[27] "Apache Mesos," Accessed: Nov. 20, 2022. [Online]. Available: https://mesos.apache.org/
[28] P. Sharma, L. Chaufournier, P. Shenoy, and Y. C. Tay, "Containers and virtual machines at scale: A comparative study," in *Proc. 17th Int. Middleware Conf.*, 2016, pp. 1–13.
[29] C. Anderson, "Docker [software engineering]," *IEEE Softw.*, vol. 32, no. 3, pp. 102–104, May/Jun. 2015.
[30] "Docker hub," Accessed: Oct. 14, 2022. [Online]. Available: https://hub.docker.com/
[31] E. Coronado, G. Cebrián-Márquez, and R. Riggio, "Enabling autonomous and connected vehicles at the 5G network edge," in *Proc. 6th IEEE Conf. Netw. Softwarization (NetSoft)*, 2020, pp. 350–352.
[32] C. Campolo, A. Iera, A. Molinaro, and G. Ruggeri, "MEC support for 5G-V2X use cases through docker containers," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, 2019, pp. 1–6.
[33] J. Tang, R. Yu, S. Liu, and J.-L. Gaudiot, "A container based edge offloading framework for autonomous driving," *IEEE Access*, vol. 8, pp. 33713–33726, 2020.
[34] C. Gómez-Huélamo et al., "360° real-time and power-efficient 3D DAMOT for autonomous driving applications," *Multimedia Tools Appl.*, vol. 81, pp. 26915–26940, Jan. 2022.

[35] S. Kugele, D. Hettler, and J. Peter, "Data-centric communication and containerization for future automotive software architectures," in *Proc. IEEE Int. Conf. Softw. Archit. (ICSA)*, 2018, p. 6509.

[36] C. Gómez-Huélamo et al., "How to build and validate a safe and reliable autonomous driving stack? A ROS based software modular architecture baseline," in *Proc. IEEE Intell. Veh. Symp. (IV)*, 2022, pp. 1282–1289.

[37] A. Mondal, P. Tigas, and Y. Gal, "Real2sim: Automatic generation of open street map towns for autonomous driving benchmarks," in *Proc. NIPS*, 2020, pp. 1–5.

[38] K. Higashiyama, K. Kimura, H. Babakarkhail, and K. Sato, "Safety and efficiency of intersections with mix of connected and non-connected vehicles," *IEEE Open J. Intell. Transp. Syst.*, vol. 1, pp. 29–34, 2020.

[39] C. Gómez-Huélamo et al., "Train here, drive there: ROS based end-to-end autonomous-driving pipeline validation in CARLA simulator using the NHTSA typology," *Multimedia Tools Appl.*, vol. 81, no. 3, pp. 4213–4240, 2022.

[40] B. Schlager, T. Goelles, M. Behmer, S. Muckenhuber, J. Payer, and D. Watzenig, "Automotive lidar and vibration: Resonance, inertial measurement unit, and effects on the point cloud," *IEEE Open J. Intell. Transp. Syst.*, vol. 3, pp. 426–434, 2022.

[41] R. Al Mallah, D. López, and B. Farooq, "Cyber-security risk assessment framework for blockchains in smart mobility," *IEEE Open J. Intell. Transp. Syst.*, vol. 2, pp. 294–311, 2021.

[42] V. Papathanasopoulou, I. Spyropoulou, H. Perakis, V. Gikas, and E. Andrikopoulou, "A data-driven model for pedestrian behavior classification and trajectory prediction," *IEEE Open J. Intell. Transp. Syst.*, vol. 3, pp. 328–339, 2022.

[43] S. Bhaskaran, "Software-in-the-loop testing for autonomous vehicles with docker," Dept. Electr. Eng. Comput. Sci., Univ. California, Berkeley, CA, USA, Rep. UCB/EECS-2022-135, 2022.

[44] J. Möller, D. Jankowski, A. Lamm, and A. Hahn, "Data management architecture for service-oriented maritime testbeds," *IEEE Open J. Intell. Transp. Syst.*, vol. 3, pp. 631–649, 2022.

[45] N. Ren, R. Jiang, and D. Zhang, "Research on CarMaker and cloud computing platform co-simulation based on APO," In *Proc. J. Phys. Conf. Series*, 2021, Art. no. 12020.

[46] D. R. Niranjan, B. C. VinayKarthik, and Mohana, "Deep learning based object detection model for autonomous driving research using CARLA simulator," in *Proc. 2nd Int. Conf. Smart Electron. Commun. (ICOSEC)*, 2021, pp. 1251–1258.

[47] R. A. Shet and S. Yao, "Cooperative driving in mixed traffic: An infrastructure-assisted approach," *IEEE Open J. Intell. Transp. Syst.*, vol. 2, pp. 429–447, 2021.

[48] C. Stadler, F. Montanari, W. Baron, C. Sippl, and A. Djanatliev, "A credibility assessment approach for scenario-based virtual testing of automated driving functions," *IEEE Open J. Intell. Transp. Syst.*, vol. 3, pp. 45–60, 2022.

[49] Y. H. Khalil and H. T. Mouftah, "LiCaNet: Further enhancement of joint perception and motion prediction based on multi-modal fusion," *IEEE Open J. Intell. Transp. Syst.*, vol. 3, pp. 222–235, 2022.

[50] B. B. Rad, H. J. Bhatti, and M. Ahmadi, "An introduction to docker and analysis of its performance," *Int. J. Comput. Sci. Netw. Security*, vol. 17, no. 3, p. 228, 2017.

[51] P. Liu et al., "Understanding the security risks of docker hub," in *Proc. Eur. Symp. Res. Comput. Security*, 2020, pp. 257–276.

[52] R. Yasrab, "Mitigating docker security issues," 2018, *arXiv:1804.05039*.

[53] "Statista." Accessed: Apr. 17, 2023. [Online]. Available: https://www.statista.com/statistics/1256245/containerization-technologies-software-market-share/

[54] "Carla." Accessed: Jan. 12, 2023. [Online]. Available: https://carla.org/

[55] "Nvidia." Accessed: Jan. 12, 2023. [Online]. Available: https://www.nvidia.com/de-de/autonomous-machines/embedded-systems/jetson-agx-xavier/

[56] "Openstreetmap." Accessed: Jan. 12, 2023. [Online]. Available: https://www.openstreetmap.de/

[57] "Blender." Accessed: Jan. 12, 2023. [Online]. Available: https://www.blender.org/

[58] "Opendrive." Accessed: Jan. 12, 2023. [Online]. Available: https://www.asam.net/standards/detail/opendrive/

[59] "Netconvert." Accessed: Jan. 12, 2023. [Online]. Available: https://sumo.dlr.de/docs/netconvert.html

[60] "Sumo." Accessed: Jan. 12, 2023. [Online]. Available: https://www.eclipse.org/sumo/

[61] Q. Li, Y. Wang, Y. Wang, and H. Zhao, "HDMapNet: An online HD map construction and evaluation framework," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, 2022, pp. 4628–4634.

[62] R. A. Kuçak, "The feature extraction from point clouds using geometric features and RANSAC algorithm," *Adv. LiDAR*, vol. 2, no. 1, pp. 15–20, 2022.

[63] J. L. Bentley, "K-D trees for semidynamic point sets," in *Proc. 6th Annu. Symp. Comput. Geomet.*, 1990, pp. 187–197.

[64] M. Muja and D. Lowe, *FLANN-Fast Library for Approximate Nearest Neighbors User Manual*, Dept. Comput. Sci., Univ. British Columbia, Vancouver, BC, Canada, 2009.

[65] J. L. Fernandez, R. Sanz, E. Paz, and C. Alonso, "Using hierarchical binary Petri nets to build robust mobile robot applications: Robograph," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2008, pp. 1372–1377.

[66] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, no. 1, pp. 3–20, 2002.

[67] H. Kellerer, U. Pferschy, and D. Pisinger, "Multidimensional knapsack problems," in *Knapsack Problems*. Heidelberg, Germany: Springer, 2004, pp. 235–283.

[68] R. A. DeVore and V. N. Temlyakov, "Some remarks on greedy algorithms," *Adv. Comput. Math.*, vol. 5, no. 1, pp. 173–187, 1996.

[69] R. Motwani and P. Raghavan, "Randomized algorithms," *ACM Comput. Surveys*, vol. 28, no. 1, pp. 33–37, 1996.

[70] "SrsLTE." Accessed: Jan. 12, 2023. [Online]. Available: https://www.srslte.com/

[71] E. Coronado, S. N. Khan, and R. Riggio, "5G-EmPOWER: A software-defined networking platform for 5G radio access networks," *IEEE Trans. Netw. Service Manag.*, vol. 16, no. 2, pp. 715–728, Jun. 2019.

[72] T. Hayashi, "Evolved packet core (EPC) network equipment for long term evolution (LTE)," *Fujitsu Sci. Tech. J*, vol. 48, no. 1, pp. 17–20, 2011.

[73] "NextEPC." Accessed: Jan. 12, 2023. [Online]. Available: https://nextepc.com/

[74] "Open vSwitch." Accessed: Jan. 12, 2023. [Online]. Available: https://www.openvswitch.org/

[75] T. O. Olwal, K. Djouani, and A. M. Kurien, "A survey of resource management toward 5G radio access networks," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 3, pp. 1656–1686, 3rd Quart., 2016.

[76] R. Molina-Masegosa, J. Gozalvez, and M. Sepulcre, "Configuration of the C-V2X mode 4 sidelink PC5 interface for vehicular communication," in *Proc. 14th Int. Conf. Mobile Ad-Hoc Sens. Netw. (MSN)*, 2018, pp. 43–48.

[77] "ETSI." Accessed: Jan. 12, 2023. [Online]. Available: https://www.etsi.org/technologies/multi-access-edge-computing

[78] A. Hegyi, H. Flinck, I. Ketyko, P. Kuure, C. Nemes, and L. Pinter, "Application orchestration in mobile edge cloud: placing of IoT applications to the edge," in *Proc. IEEE 1st Int. Workshops Found. Appl. Self* Syst. (FAS* W)*, 2016, pp. 230–235.

[79] "Circles." Accessed: Jan. 12, 2023. [Online]. Available: https://circles-consortium.github.io

[80] R. C. Coulter, "Implementation of the pure pursuit path tracking algorithm," Robot. Inst., Univ. Carnegie-Mellon, Pittsburgh, PA, USA, Rep. CMU-RI-TR-92-01, 1992.

[81] K. Rana and M. Zaveri, "A-star algorithm for energy efficient routing in wireless sensor network," in *Trends in Network and Communications*. Heidelberg, Germany: Springer, 2011, pp. 232–241.

[82] J. A. Benayas, J. L. Fernández, R. Sanz, and A. R. Diéguez, "The beam-curvature method: A new approach for improving local realtime obstacle avoidance," *IFAC Proc. Vol.*, vol. 35, no. 1, pp. 409–414, 2002.

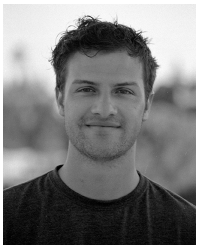
[83] E. Romera, J. M. Álvarez, L. M. Bergasa, and R. Arroyo, "ERFNet: Efficient residual factorized convnet for real-time semantic segmentation," *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 1, pp. 263–272, Jan. 2018.

[84] D. Held, J. Levinson, and S. Thrun, "Precision tracking with sparse 3D and dense color 2D data," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2013, pp. 1138–1145.

[85] R. E. Kalman, "A new approach to linear filtering and prediction problems," *J. Basic Eng.*, vol. 82, no. 1, pp. 35–45, Mar. 1960. 1960.

[86] J. H. Friedman, F. Baskett, and L. J. Shustek, "An algorithm for finding nearest neighbors," *IEEE Trans. Comput.*, vol. 100, no. 10, pp. 1000–1006, Oct. 1975.
[87] A. M. Joy, "Performance comparison between Linux containers and virtual machines," in *Proc. Int. Conf. Adv. Comput. Eng. Appl.*, 2015, pp. 342–346.
[88] R. K. Barik, R. K. Lenka, K. R. Rao, and D. Ghose, "Performance analysis of virtual machines and containers in cloud computing," in *Proc. Int. Conf. Comput., Commun. Autom. (ICCCA)*, 2016, pp. 1204–1210.
[89] K. Ren, Q. Wang, C. Wang, Z. Qin, and X. Lin, "The security of autonomous driving: Threats, defenses, and future directions," *Proc. IEEE*, vol. 108, no. 2, pp. 357–372, Feb. 2020.
[90] D. Krajzewicz, L. Bieker, and J. Erdmann, "Preparing simulative evaluation of the GLOSA application," in *Proc. CD ROM 19th ITS World Congr.*, 2012, Art. no. EU-00630.
[91] M. Y. I. Idris, Y. Y. Leng, E. M. Tamil, N. M. Noor, and Z. Razak, "Car park system: A review of smart parking system and its technology," *Inf. Technol. J.*, vol. 8, no. 2, pp. 101–113, 2009.
[92] J. B. Greenblatt and S. Shaheen, "Automated vehicles, on-demand mobility, and environmental impacts," *Current Sustain. Renew. Energy Rep.*, vol. 2, no. 3, pp. 74–81, 2015.
[93] F. Caicedo, C. Blazquez, and P. Miranda, "Prediction of parking space availability in real time," *Expert Syst. Appl.*, vol. 39, no. 8, pp. 7281–7290, 2012.
[94] D. M. Gavrila and V. Philomin, "Real-time object detection for 'smart' vehicles," in *Proc. 11th IEEE Int. Conf. Comput. Vis.*, vol. 1, 1999, pp. 87–93.
[95] "Diginet-PS." Accessed: Oct. 15, 2022. [Online]. Available: https://diginet-ps.de/en/home/
[96] "Lanelet2." Accessed: Apr. 17, 2023. [Online]. Available: https://github.com/fzi-forschungszentrum-informatik/Lanelet2
[97] S. Ma, S. Guo, K. Wang, and M. Guo, "Service demand prediction with incomplete historical data," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, 2019, pp. 912–922.

BENJAMIN ACAR received the master's degree in technomathematics (mathematics with a minor in physics) from the Karlsruhe Institute of Technology, Germany. He is currently pursuing the Ph.D. degree in computer science with Technische Universität Berlin, focusing on multiagent systems. Previously, he worked as a Risk Analyst in one of the largest German banks. His research interests encompass distributed systems, machine learning, and software engineering.



MARC GUERREIRO AUGUSTO is a computer scientist who has been working as a consultant in the port logistics industry with an emphasis on process optimization and automation. He currently leads the BeIntelli Research Project, exploring AI in mobility based on platform economy, a unique showcase project on autonomous driving in the heart of Berlin. He also acts as Partner and a Program Manager with the Center for Tangible AI and Digitalization (ZEKI). His research focuses on platform economy and distributed AI for CCAM solutions.

MARIUS STERLING received the master's degrees in statistics from the Joint Masters Program in Berlin, a cooperation between Humboldt-, Technische, and Freie Universität Berlin, and in statistics for smart data from the École Nationale de la Statistique et de l'Analyse de l'Information, Rennes, France. He currently leads the sub-project responsible for platform development and creating selected showcase services within BeIntelli Research Project. His research interests encompass data science, applied machine and deep learning, and CCAM, with a particular interest in CCAM-based smart and efficient parking systems.

FIKRET SIVRIKAYA received the bachelor's degree in computer engineering from Bogazici University, Istanbul, Turkey, in 2000, and the Ph.D. degree in computer science from Rensselaer Polytechnic Institute, NY, USA, in 2007. Since 2008, he has been working as a Senior Researcher and a Lecturer with Technische Universität Berlin, Germany. He has also been serving as the Research Director of the German-Turkish Advanced Research Center for ICT, an affiliated institute of TU Berlin, Berlin, Germany, since 2016. His research interests include future mobile networks, Internet of Things, and artificial intelligence, with an application focus on intelligent transport systems and smart cities.

SAHIN ALBAYRAK received the Ph.D. degree in computer science and the Habilitation degree from Technische Universität Berlin, Germany, in 1992 and 2002, respectively. He currently holds the Chair Agent Technologies in Business Applications and Telecommunication, TU Berlin as a Full Professor, where he is the Founder and the Head of the Distributed Artificial Intelligence Laboratory. He is also the Founding Director of the Connected Living Association, the German-Turkish Advanced Research Centre for ICT, and the Center for Tangible AI and Digitalization (ZEKI), Berlin, Germany. His research interests include distributed systems, machine learning, cybersecurity, multiagent systems, and autonomous systems, with their particular applications in autonomous driving, smart cities, smart energy systems, telecommunications, and preventive health.